Argonne NATIONAL LABORATORY

Casper

An Asynchronous Progress Model for MPI RMA on Many-Core Architectures

Min Si

Guest graduate student at Argonne National Laboratory, IL, USA

Mentor : Antonio J. Peña Supervisor : Pavan Balaji

PhD student at University of Tokyo, Tokyo, Japan

Advisor : Yutaka Ishikawa

Download slides: http://sudalab.is.s.u-tokyo.ac.jp/~msi/pdf/casper-seminar-20150423.pdf

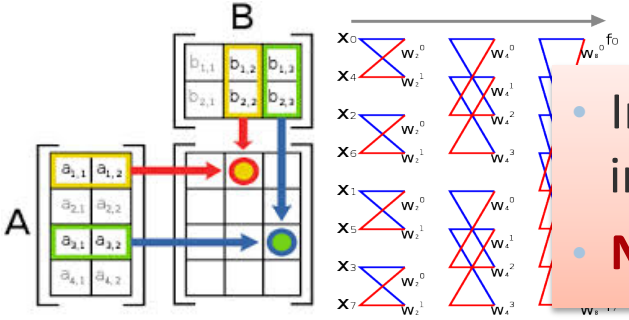
THE UNIVERSITY OF TOKYO

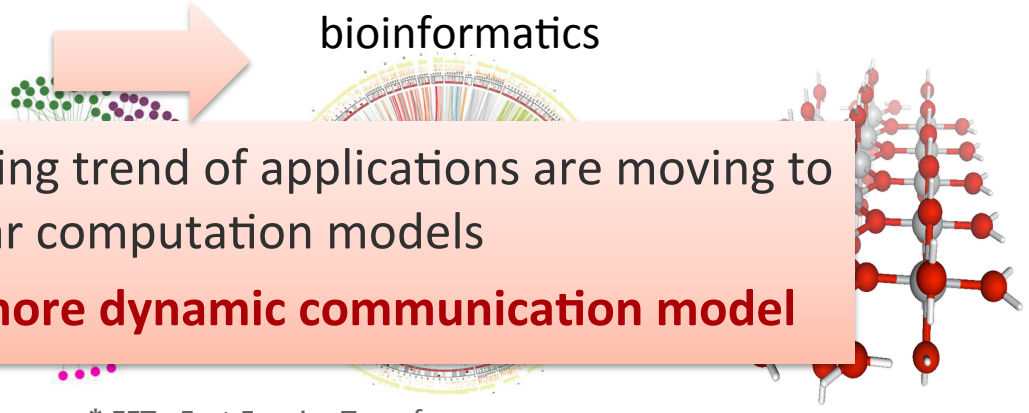
U.S. DEPARTMENT OF ENERGY

Irregular Computations

- **Regular computations**
  - Organized around dense vectors or matrices
  - **Regular data movement** pattern, use **MPI SEND/RECV or collectives**
  - More local computation, less data movement
  - Example: stencil computation, matrix multiplication, FFT*

- **Irregular computations**
  - Organized around graphs, sparse vectors, more "data driven" in nature
  - Data movement pattern is **irregular and data-dependent**
  - **Growth rate of data movement is much faster than computation**
  - Example: social network analysis, bioinformatics





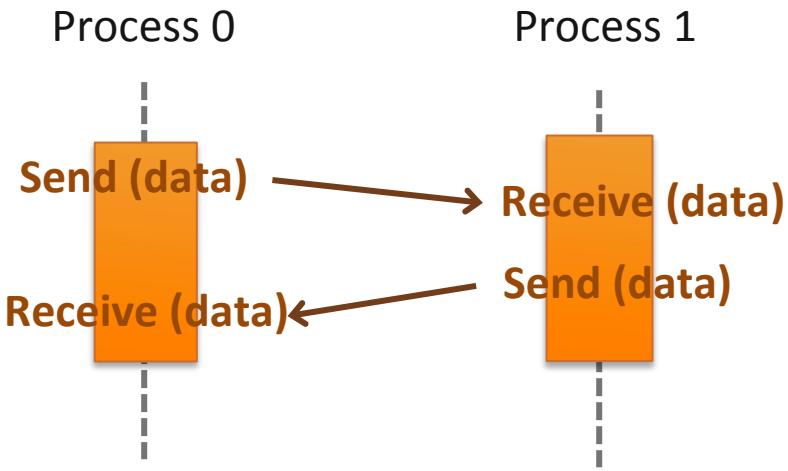
- Increasing trend of applications are moving to irregular computation models
- **Need more dynamic communication model**

* **FFT** : Fast Fourier Transform

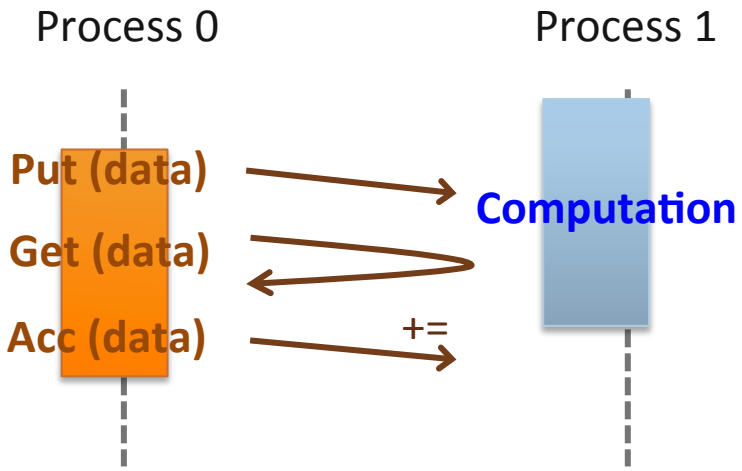
* The primary contents of this slide are contributed by Xin Zhao.

Message Passing Models

- **Two-sided communication**



- **One-sided communication (Remote Memory Access)**



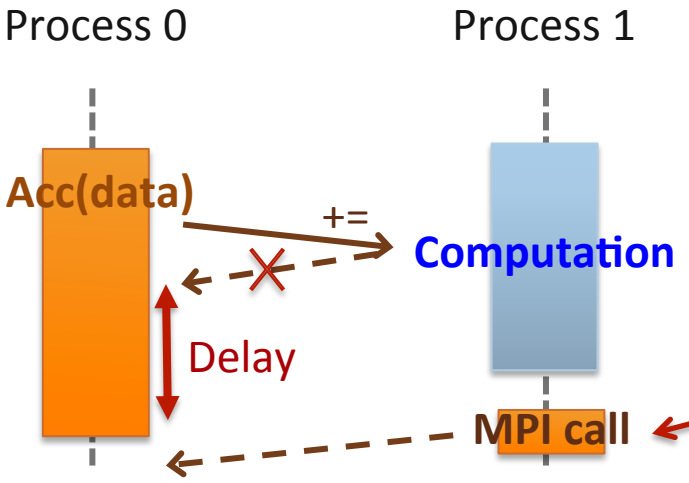
Feature:

- Origin (P0) specifies all communication parameters
- Target (P1) does not explicitly receive or process message

Is communication always asynchronous ?

Problems in Asynchronous Progress

- **One-sided operations are not truly one-sided**
  - In most platforms (e.g., InfiniBand, Cray)
    - Some operations are hardware supported (e.g., contiguous PUT/GET)
    - Other operations **have to be done in software** (e.g., 3D accumulates of double precision data)



Software implementation of one-sided operations means that **the target process has to make an MPI call to make progress**.

Not TRULY asynchronous !

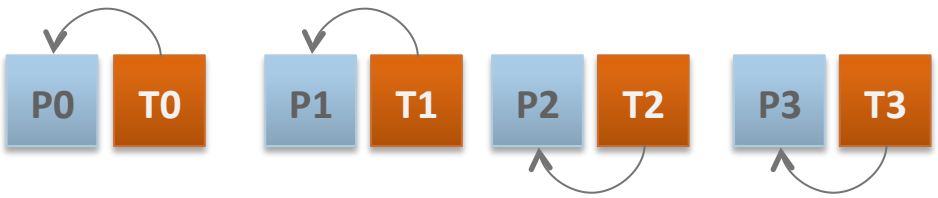
Traditional Approach of ASYNC Progress (1)

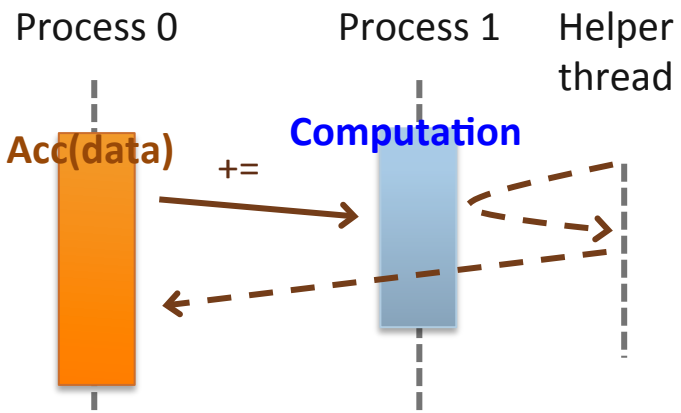
- **Thread-based approach**
  - Every MPI process has a **communication dedicated background thread**
  - Background thread polls MPI progress in order to handle incoming messages for this process
  - Example: MPICH default asynchronous thread, SWAP-bioinformatics

Cons:

× **Waste half of computing cores** or **oversubscribe** cores

× Overhead of **Multithreading safety** of MPI



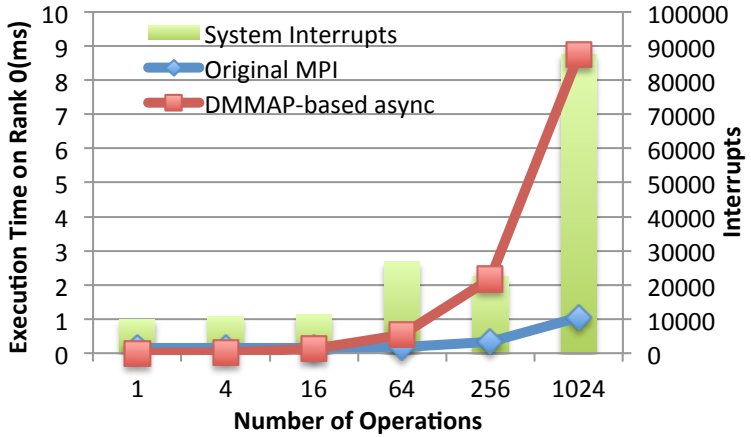


Traditional Approach of ASYNC Progress (2)

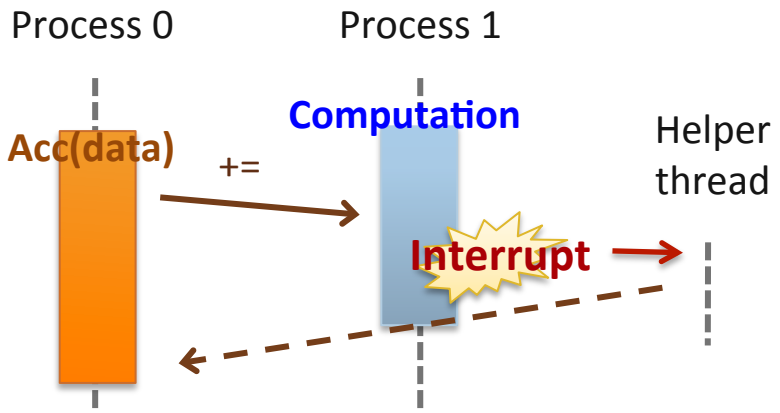
- **Interrupt-based approach**
  - Assume all hardware resources are busy with user computation on target processes
  - Utilize **hardware interrupts** to awaken a kernel thread and process the incoming RMA messages
  - i.e., DMMAP based Cray MPI, IBM MPI on Blue Gene/P

Cons:

× Overhead of **frequent interrupts**

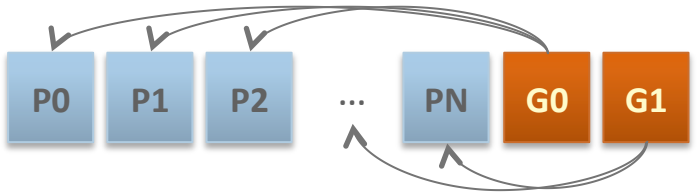


DMMAP-based ASYNC overhead on Cray XC30



Casper Process-based ASYNC Progress

- **Multi- and many-core architectures**
  - Rapidly growing number of cores
  - **Not all of the cores are always keeping busy**



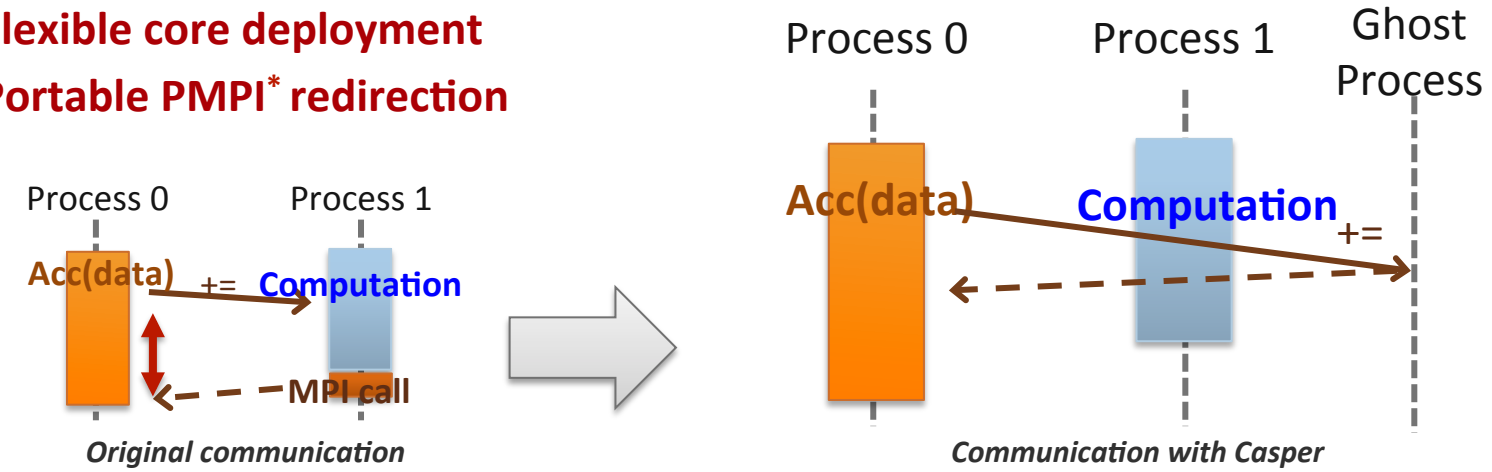
- **Process-based asynchronous progress**
  - Dedicating **arbitrary number of cores to "ghost processes"**
  - **Ghost process intercepts all RMA operations** to the user processes

Pros:

✓ No overhead caused by **multithreading safety** or **frequent interrupts**

✓ **Flexible core deployment**

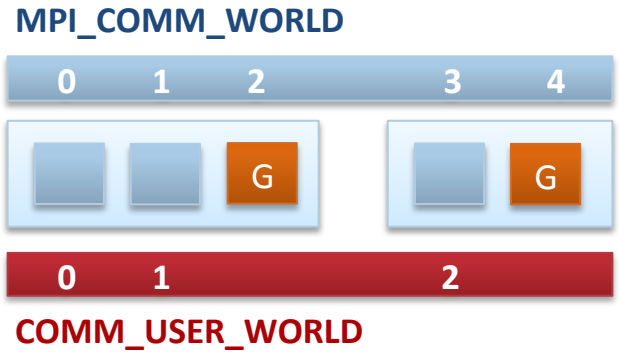
✓ **Portable PMPI* redirection**

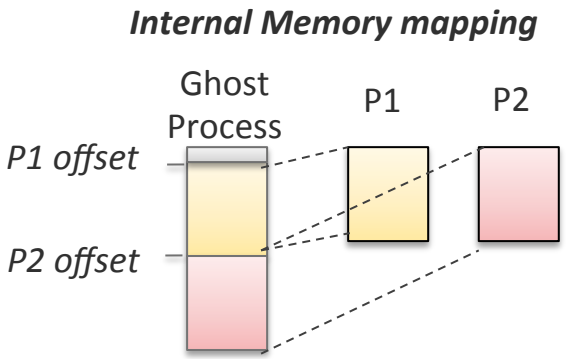


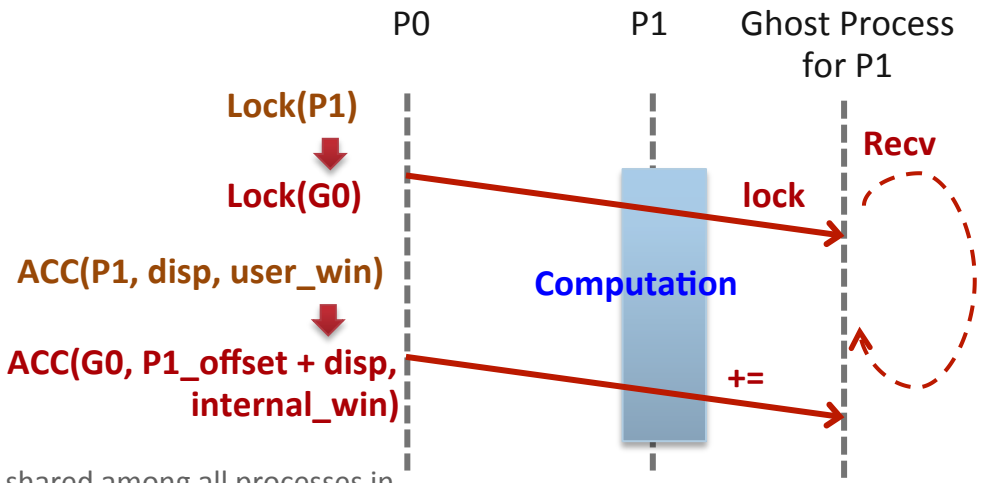
* **PMPI** : name-shifted profiling interface of MPI

Basic Design of Casper

- **Three primary functionalities**
  1. Transparently replace MPI_COMM_WORLD by **COMM_USER_WORLD**
  2. **Shared memory mapping** between local user and ghost processes by using MPI-3 MPI_Win_allocate_shared*
  3. **Redirect RMA operations** to ghost processes







* **MPI_WIN_ALLOCATE_SHARED** : Allocates window that is shared among all processes in the window's group, usually specified with MPI_COMM_TYPE_SHARED communicator.

Outline

- Ensuring Correctness and Performance
- Evaluation
- Asynchronous Progress Runtime Adaptation
- Next Steps

[1] M. Si, A. J. Pena, J. Hammond, P. Balaji, M. Takagi, and Y. Ishikawa, “Casper: An asynchronous progress model for MPI RMA on many-core architectures,” in Parallel and Distributed Processing (IPDPS), 2015.

[2] M. Si, A. J. Pena, J. Hammond, P. Balaji, and Y. Ishikawa. Scaling NWChem with Efficient and Portable Asynchronous Communication in MPI RMA. In Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium, 2015. (Accepted)

Ensuring Correctness and Performance

Correctness challenges

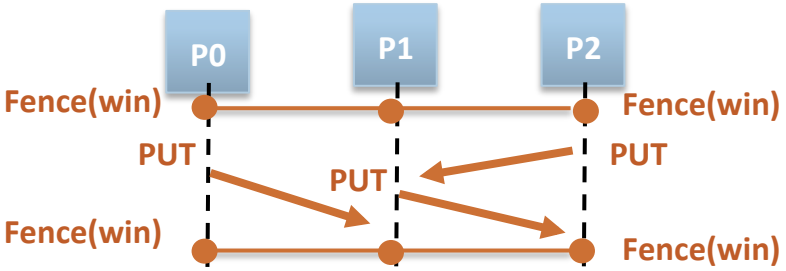
1. Lock Permission Management
2. Self Lock Consistency
3. Managing Multiple Ghost Processes
4. Multiple Simultaneous Epochs

Performance challenge

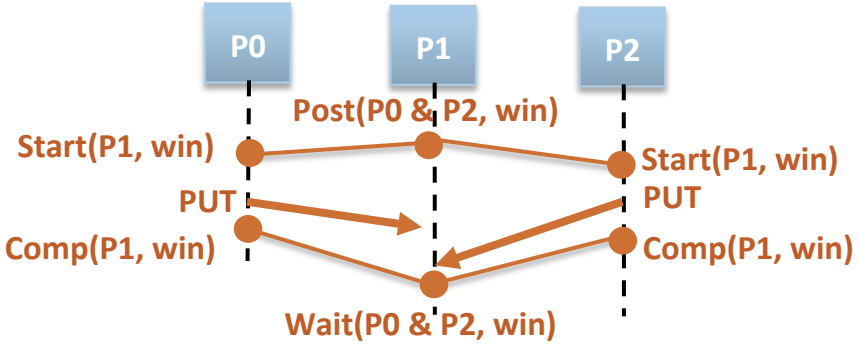
1. Memory Locality

RMA synchronization modes

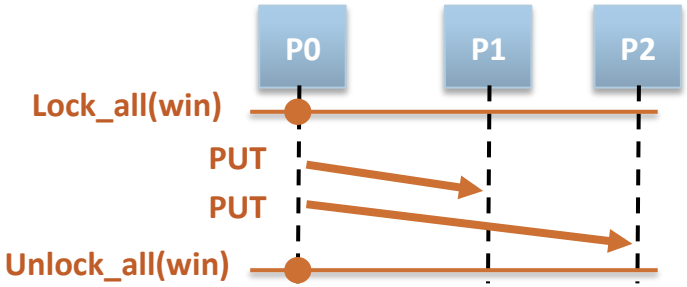
- **Active-target mode**
  - Both origin and target issue synchronization
  - **Fence** (like a global barrier)



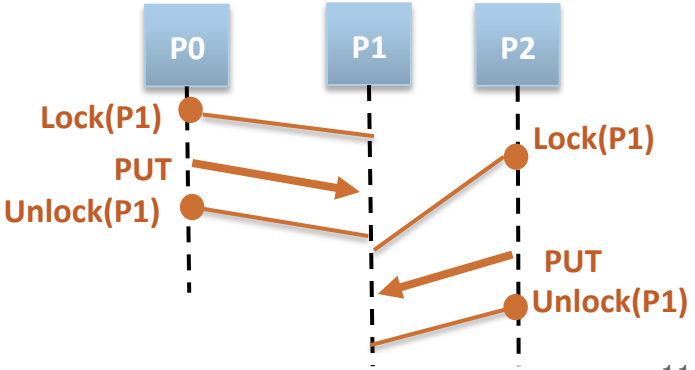
  - **PSCW** (subgroup of Fence)



- **Passive-target mode**
  - Only origin issues synchronization
  - **Lock_all** (shared)



  - **Lock** (shared or exclusive)

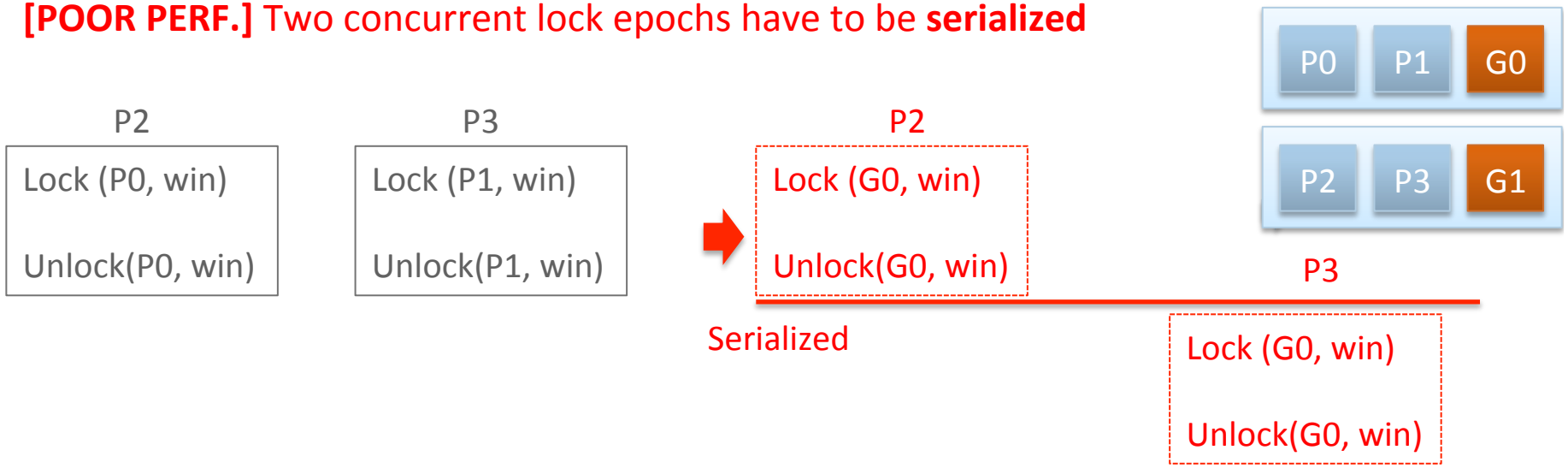


[Correctness Challenge 1]

Lock Permission Management for Shared Ghost Processes (1)

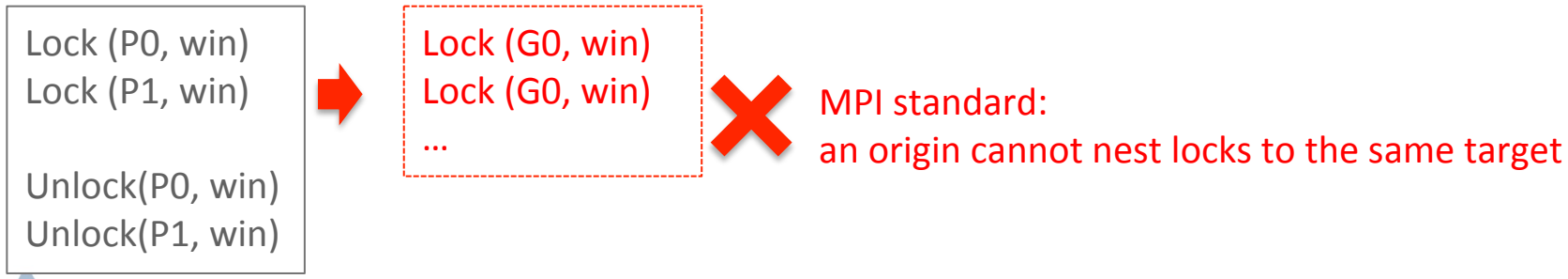
1. Two origins access two targets sharing the same ghost process

[POOR PERF.] Two concurrent lock epochs have to be **serialized**



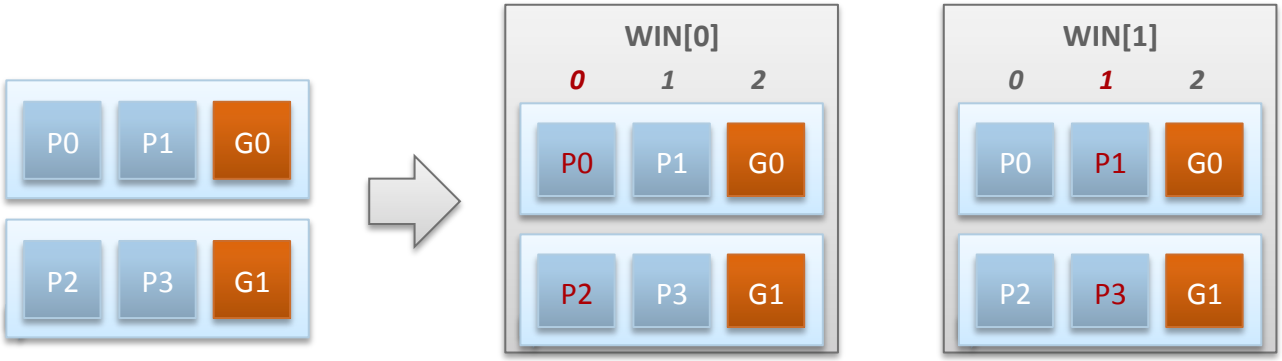
2. An origin accesses two targets sharing the same ghost process

[INCORRECT] Nested locks to the same target



[Correctness Challenge 1]
Lock Permission Management for Shared Ghost Processes (2)

- **Solution**
  - **N Windows**
    - N = max number of processes on every node
    - COMM. to **i_{th} user process on each node** goes to **i_{th} window**



- **User hint optimization**
  - Window info "**epochs_used**" (fence|pscw|lock|lockall by default)
    - **If "epochs_used" contains "lock", create N windows**
    - Otherwise, only create a single window

[Correctness Challenge 2] Self Lock Consistency (1)

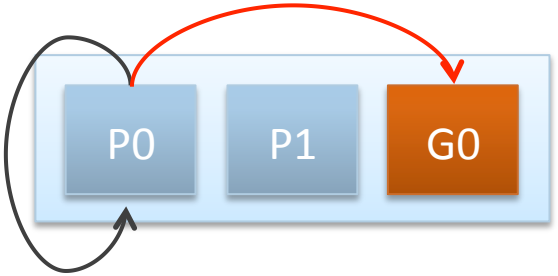
P0

```
Lock (P0, win)

x=1
y=2
…

Unlock(P0, win)
```

MPI standard:
Local lock must be acquired immediately



```
Lock (G0, win)

Unlock(G0, win)
```

MPI standard:
Remote lock may be delayed..

[Correctness Challenge 2] Self Lock Consistency (2)

- **Solution (2 steps)**
  1. **Force-lock with HIDDEN BYTES***

```
Lock (G0, win)
Get (G0, win)
Flush (G0, win)
```
// Lock is acquired

  2. **Lock self**

```
Lock (P0, win)
```
// memory barrier for managing
// memory consistency

- **User hint optimization**
  - Window info **no_local_loadstore**
    - Do not need both 2 steps
  - Epoch assert **MPI_MODE_NOCHECK**
    - Only need the 2_{nd} step

* MPI standard defines **unnecessary restriction on concurrent GET and accumulate**. See MPI Standard Version 3.0 , page page 456, line 39.

[Correctness Challenge 3]
Managing Multiple Ghost Processes (1)

1. Lock permission among multiple ghost processes

[INCORRECT] Two **EXCLUSIVE locks** to the same target may be **concurrently acquired**

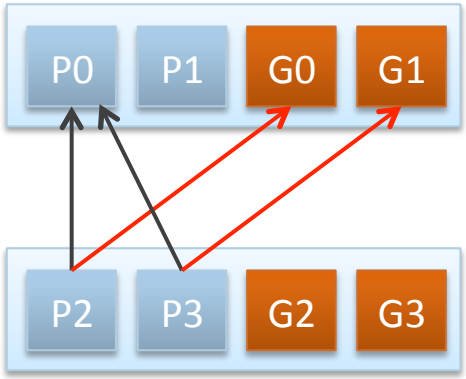
P2

```
Lock (EXCLUSIVE, P0, win)
PUT(P0)
Unlock(P0, win)
```

P3

Serialized

```
Lock (EXCLUSIVE, P0, win)
PUT(P0)
Unlock(P0, win)
```



P2

```
Lock (EXCLUSIVE, G0, win)
Lock (EXCLUSIVE, G1, win)

// get G0
G = randomly_pick_ghost();
PUT(G)
...
```

P3

```
Lock (EXCLUSIVE, G0, win)
Lock (EXCLUSIVE, G1, win)

// get G1
G = randomly_pick_ghost();
PUT(G)
...
```

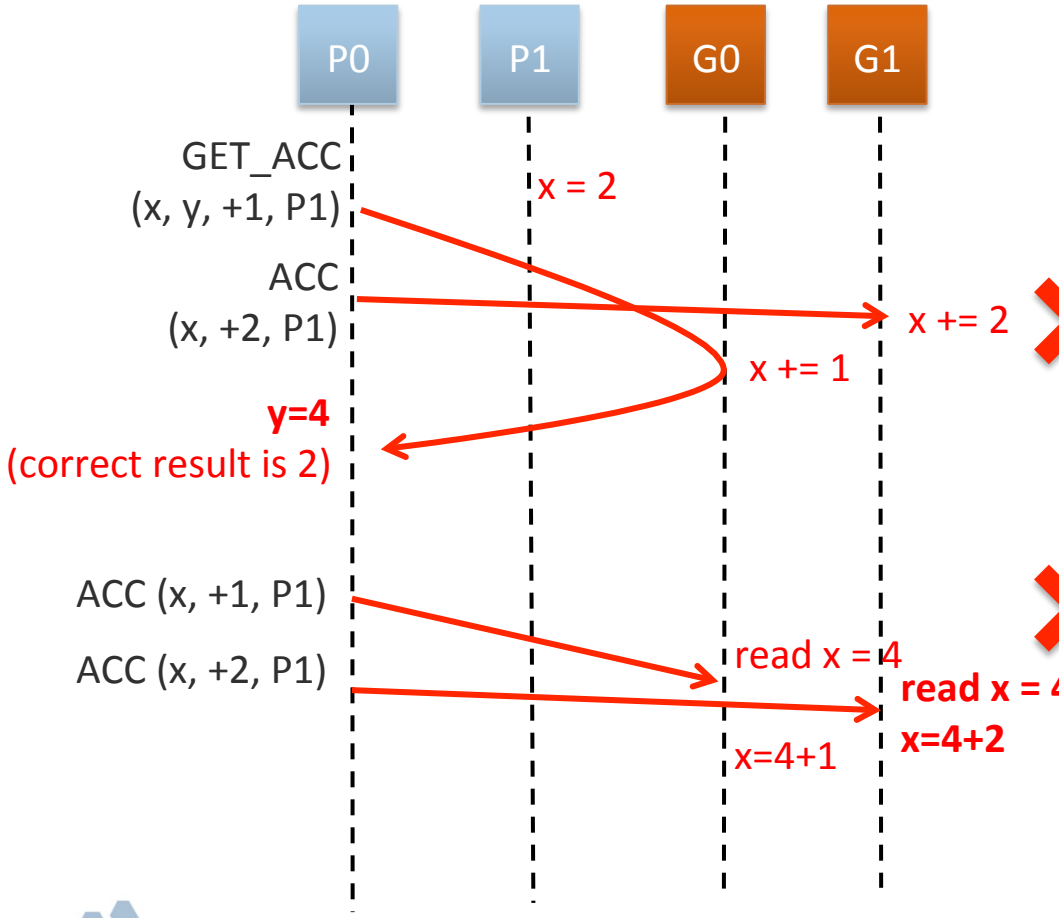
Empty lock can be ignored,
P2 and P3 may concurrently acquire lock on G0 and G1

[Correctness Challenge 3]

Managing Multiple Ghost Processes (2)

2. Ordering and Atomicity constraints for Accumulate operations

[INCORRECT] Ordering and Atomicity cannot be maintained by MPI among multiple ghost processes

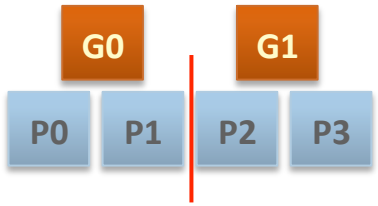


MPI standard:
Same origin && same target location accumulates **must be ordered**

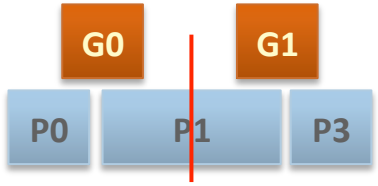
MPI standard:
Concurrent accumulates to the same target location **must be atomic per basic datatype element.**

[Correctness Challenge 3]
Managing Multiple Ghost Processes (3)

- **Solution (2 phases)**
  1. **Static-Binding Phase**
     - **Rank binding model**
       - **Each user process** binds to a single ghost process
     - **Segment binding model**
       - **Segment total exposed memory** on each node **into N_G chunks**
       - **Each chunk** binds to a single ghost process
     - Only redirect RMA operations to the bound ghost process
     - Fixed lock and ACC ordering & atomicity issues
     - But **only suitable for balanced communication patterns**



Static-rank-binding



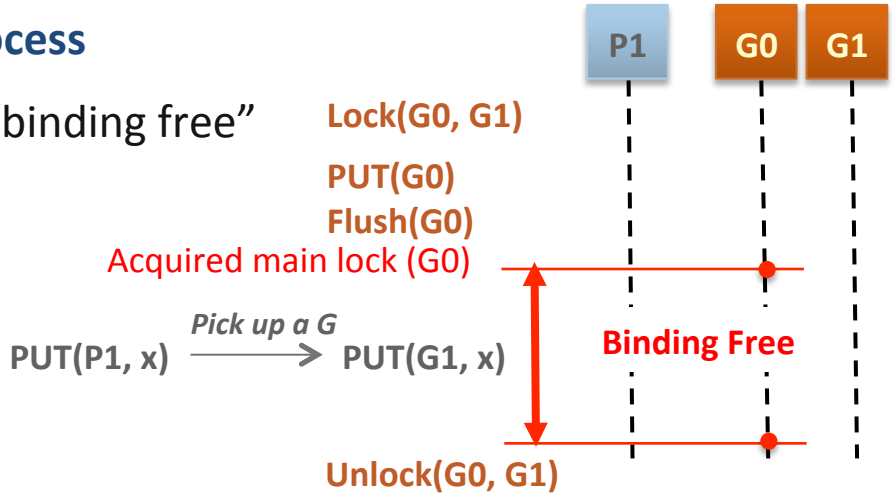
Static-segment-binding

[Correctness Challenge 3]
Managing Multiple Ghost Processes (4)

1. Static-Binding Phase

   Optimization for dynamic communication patterns

2. **Static-Binding-Free Phase**
   - **After operation + flush issued**, "main lock" is acquired
   - **Dynamically select target ghost process**
   - Accumulate operations can not be "binding free"

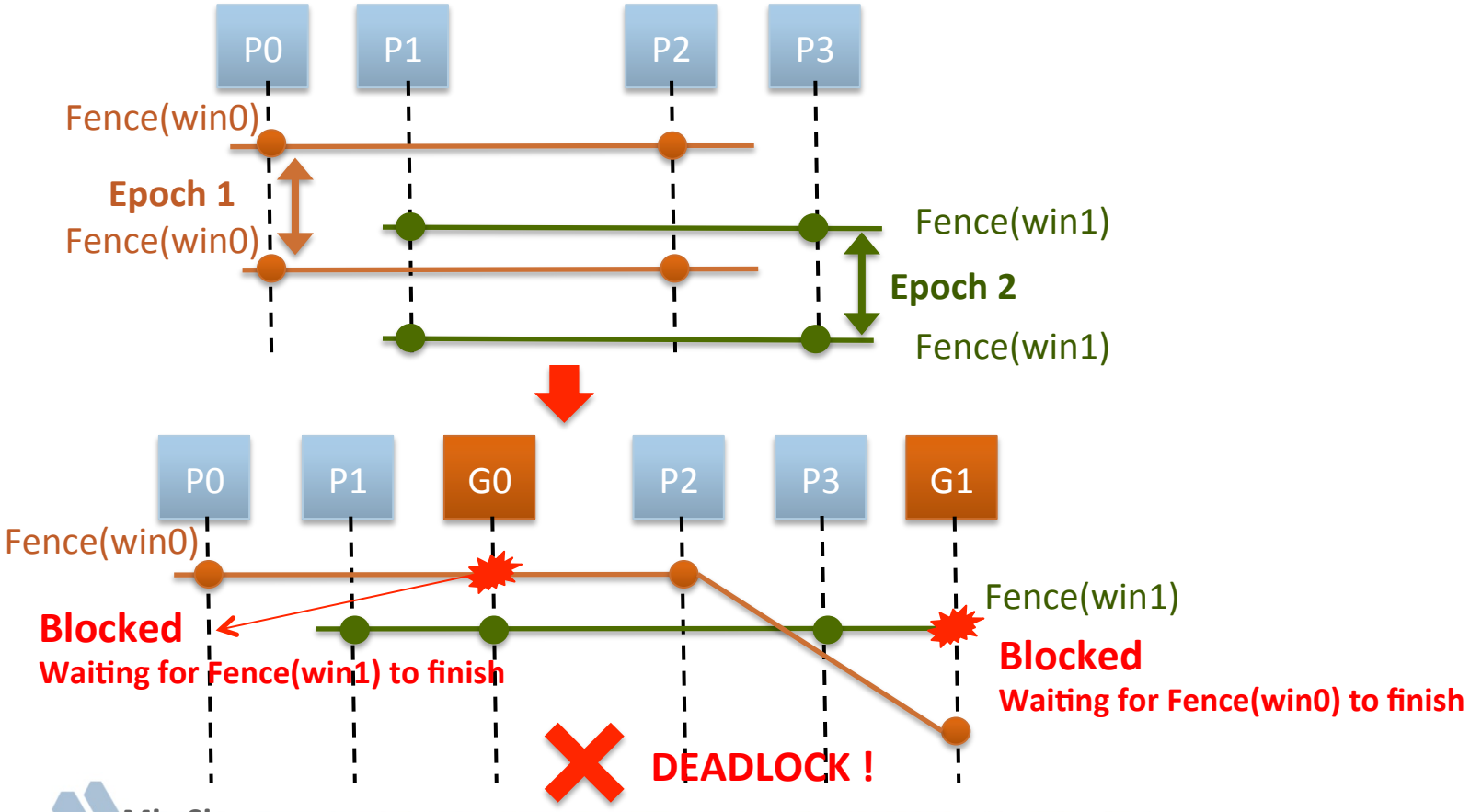


[Correctness Challenge 4]

Multiple Simultaneous Epochs – Active Epochs (1)

- **Simultaneous fence epochs on disjoint sets of processes sharing the same ghost processes**

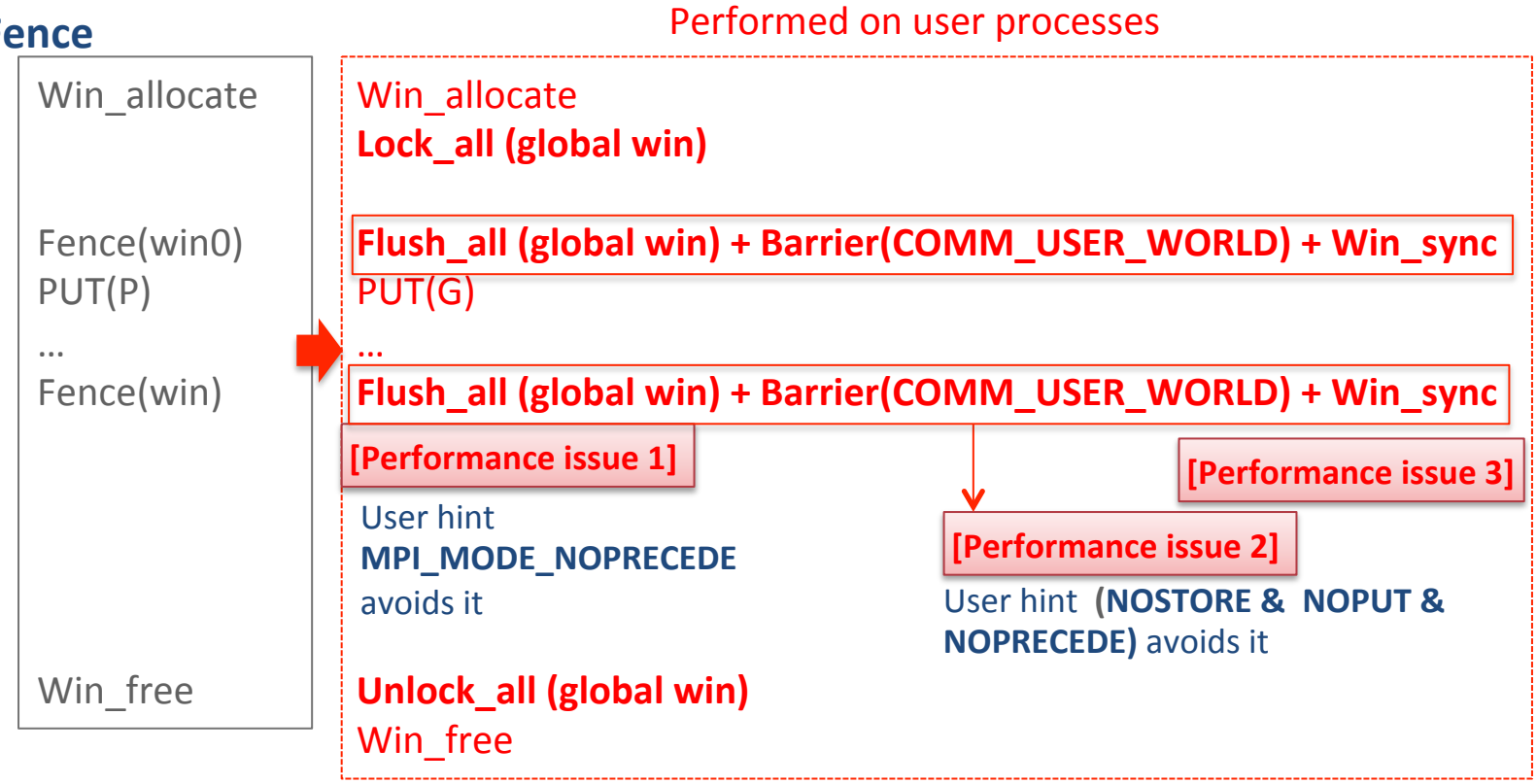
  [INCORRECT] Deadlock !



[Correctness Challenge 4]

Multiple Simultaneous Epochs - Active Epochs (2)

- **Solution**
  - Every user window has an **internal "global window"**
  - **Translate to passive-target mode**
  - **Fence**

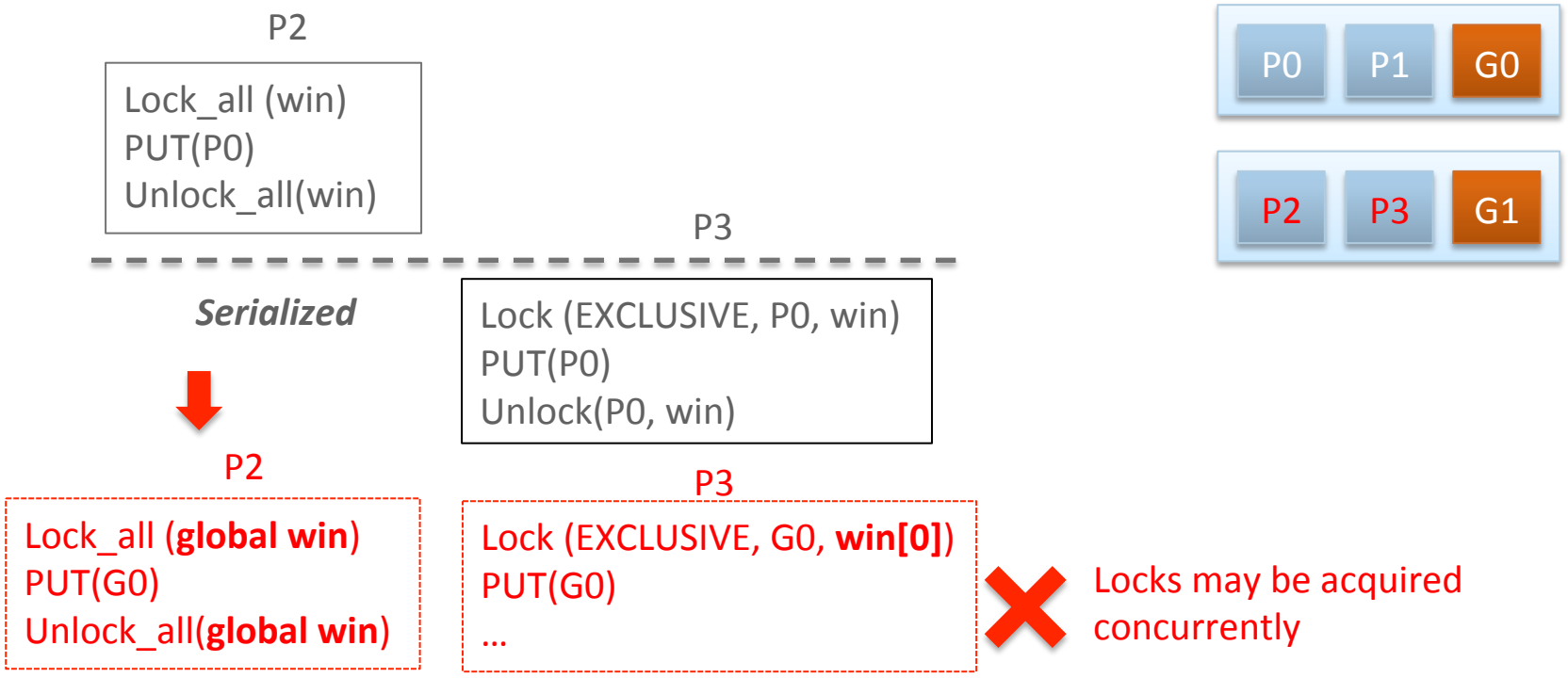


  - **PSCW ➡ Flush + Send-Receive**

[Correctness Challenge 4]
Multiple Simultaneous Epochs – Lock_all (1)

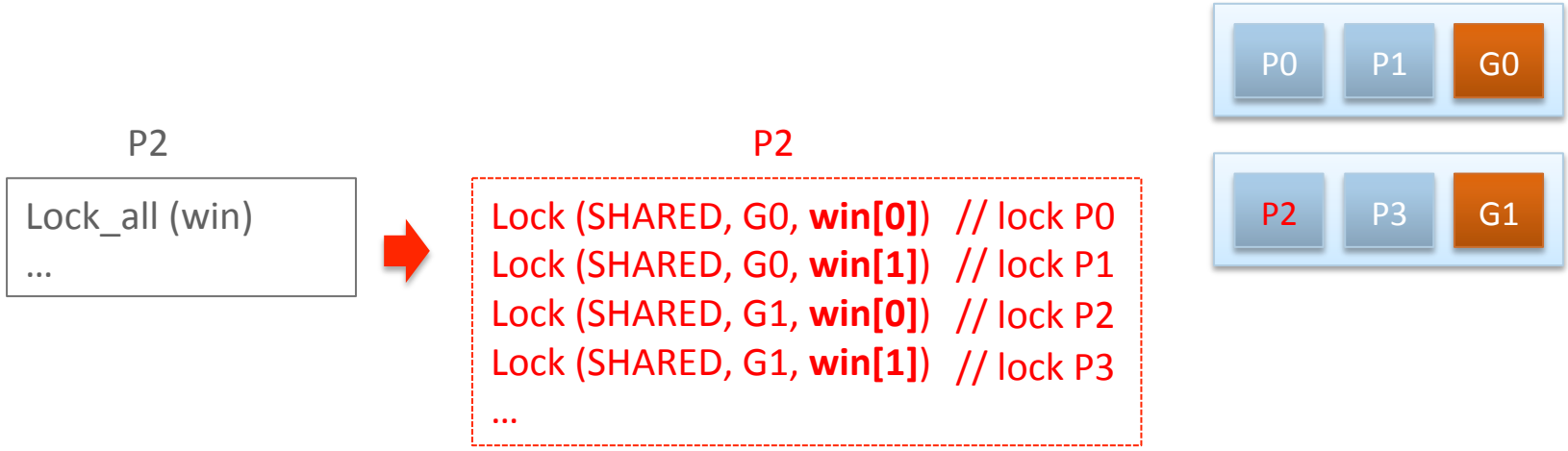
- **Lock_all only**
  - **Same translation as that for Fence**
    - **lock_all in win_allocate, flush_all in unlock_all**

[INCORRECT] Lock_all and EXCLUSIVE lock on the same window may be **concurrently acquired**



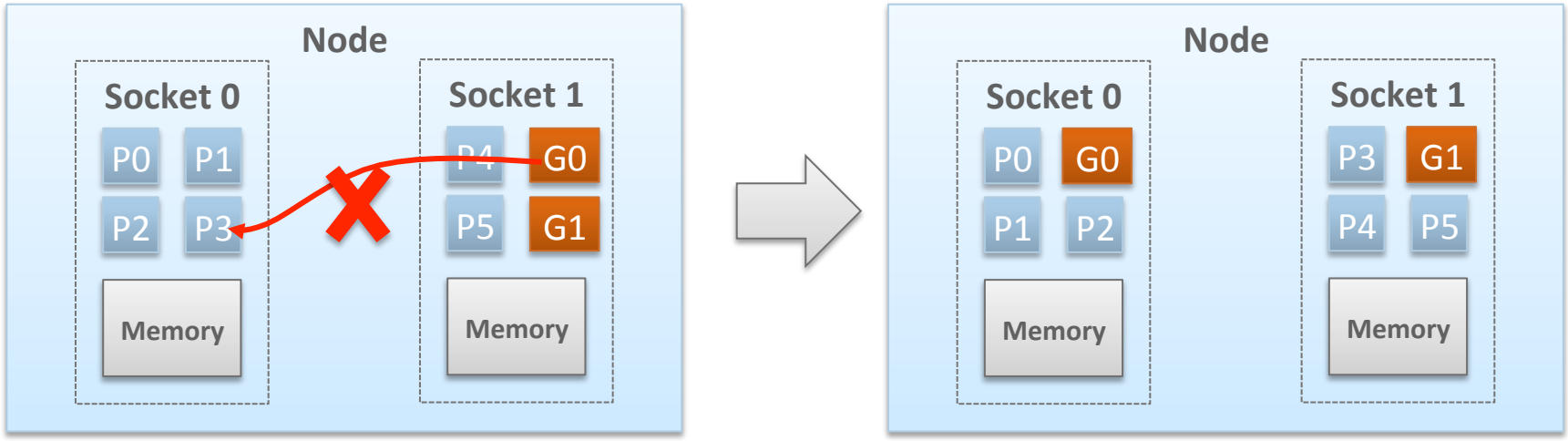
[Correctness Challenge 3]
Multiple Simultaneous Epochs – Lock_all (2)

- **Solution**
  - **Translate lock_all to a series of locks to all ghost processes**



[Performance Challenge] Memory Locality

- Casper internally detects the location of the user processes
- Only bind the **closest ghost processes**
- i.e., P0-2 are bound to G0, P3-5 are bound to G1



Evaluation

Asynchronous Progress Microbenchmark

NWChem Quantum Chemistry Application

Experimental Resources

- Experiment platform
  1. NERSC Edison Cray XC30*
     - 24 cores per node
     - Cray MPI v6.3.1

RMA implementation

	HW-handled OP	SW-handed OP	ASYNC. mode
Original mode	NONE	All	Thread
DMAPP mode	Contig. PUT/GET	Noncontig OP, ACC	Interrupt

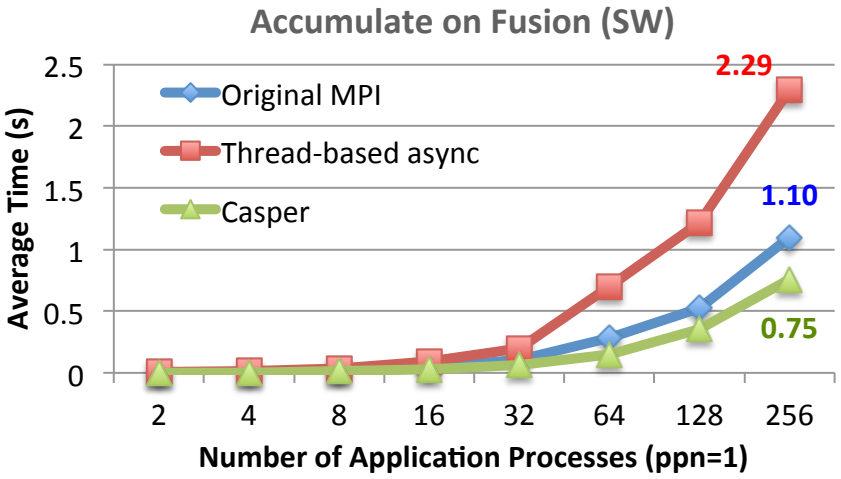
  2. Fusion cluster
     - 8 cores per node
     - MVAPICH v2.0rc11

RMA implementation

HW-handled OP	SW-handed OP	ASYNC. mode
Contig. PUT/GET	Noncontig OP, ACC	Thread

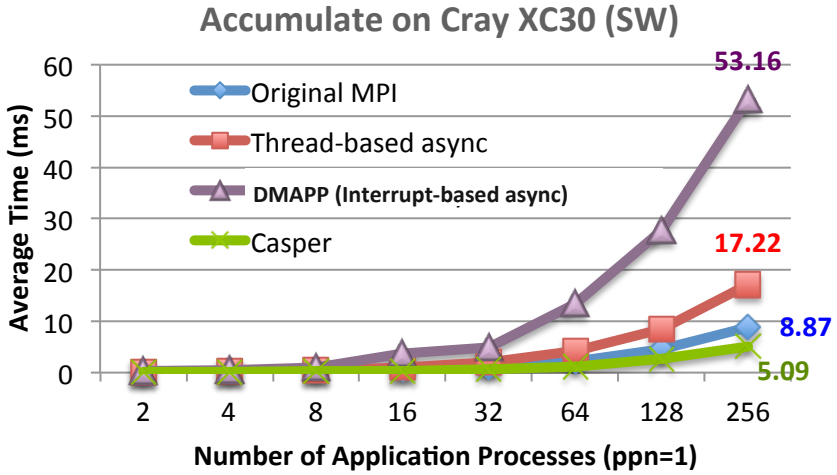
* https://www.nersc.gov/users/computational-systems/edison/configuration/

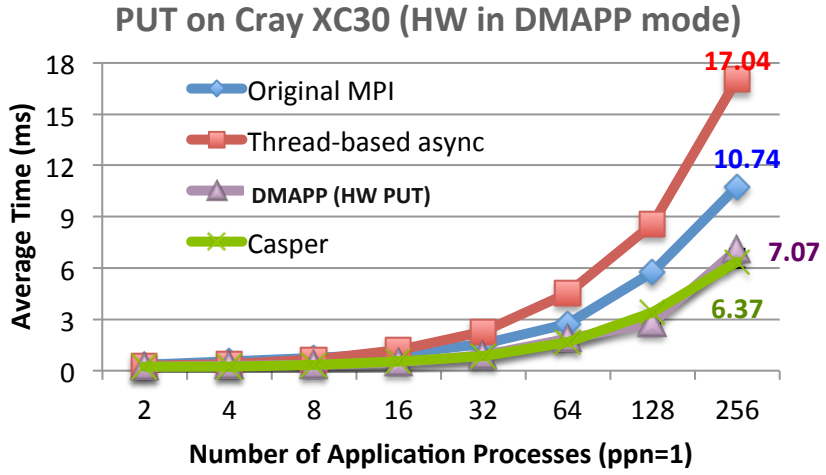
Evaluation 1. Asynchronous Progress Microbenchmark



Test scenario

```
Lock_all (win);
for (i=0; i<nproc; i++) {
        OP(i, double, cnt = 1);
        Flush(i);
        busy wait 100us;
}
Unlock_all (win)
```



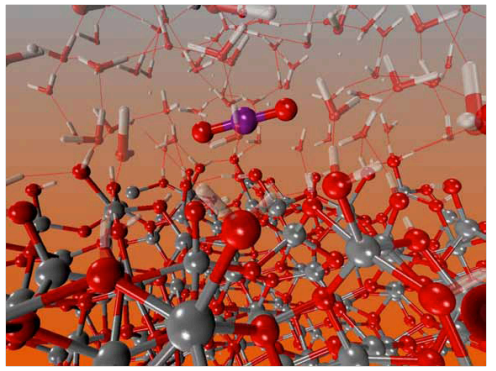


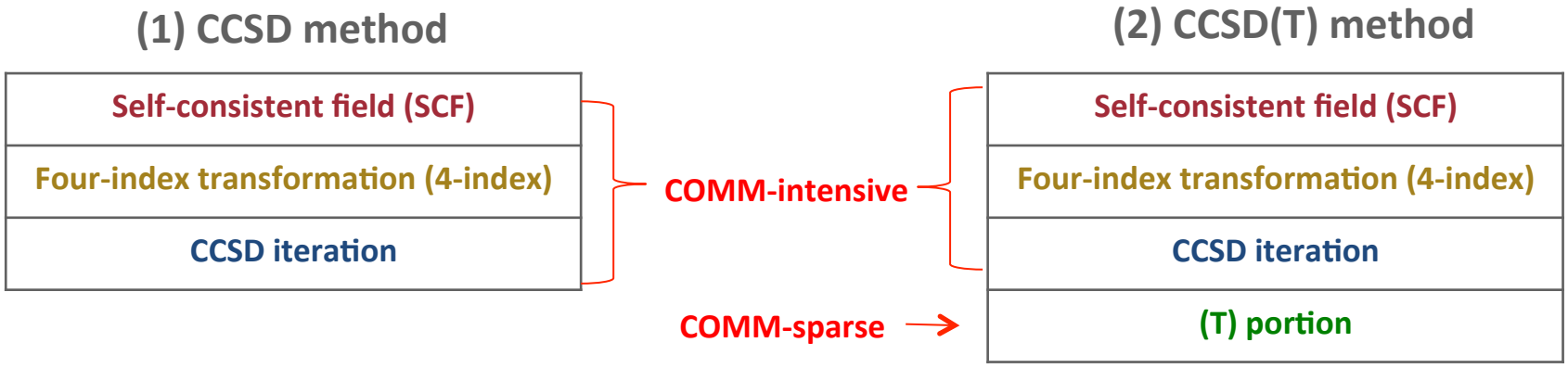
Casper provides asynchronous progress for SW-handled operations.

Casper performs the same performance as that of HW operations

Evaluation 2. NWChem Quantum Chemistry Application (1)

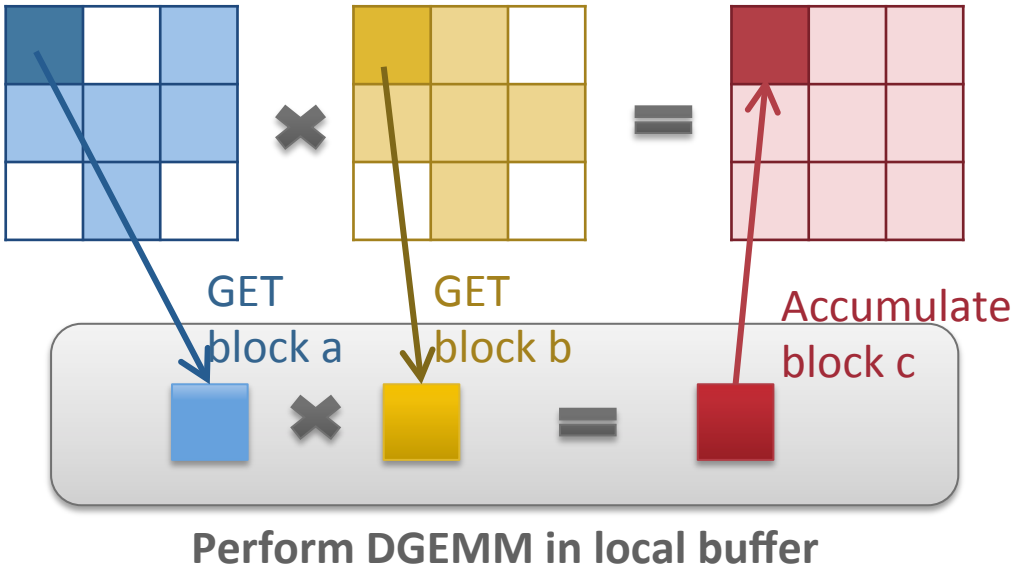
- Computational chemistry application suite composed of many types of simulation capabilities.
- **ARMCI-MPI** (Portable implementation of **Global Arrays over MPI RMA**)
- Focus on most common used **CC (coupled-cluster) simulations**





Evaluation 2. NWChem Quantum Chemistry Application (2)

- Typical computation-communication pattern
  - **Get-Compute-Update**

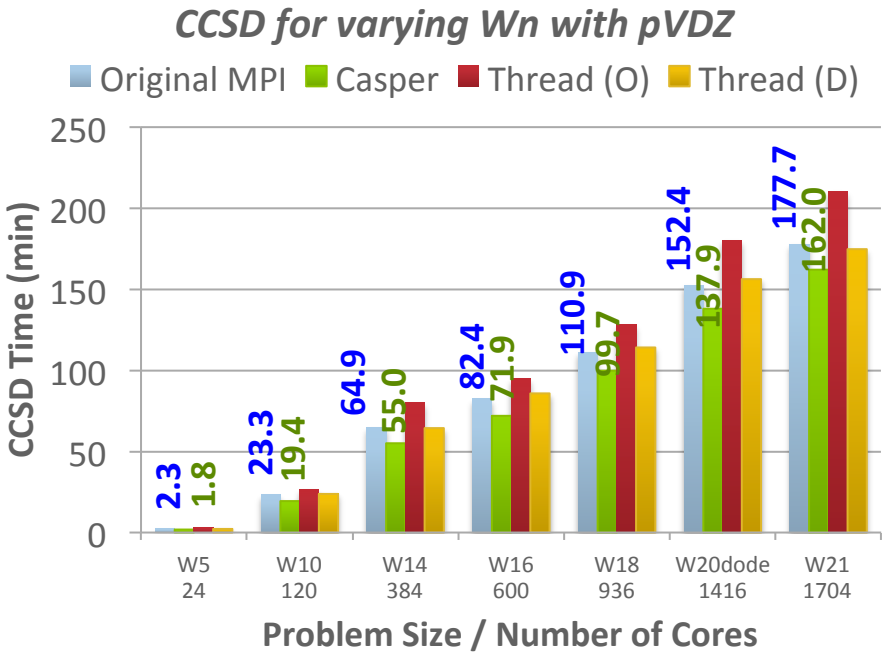



```
for i in I blocks:
  for j in J blocks:
    for k in K blocks:
      GET block a from A
      GET block b from B
      c += a * b        Heavy computation
    end do
    ACC block c to C
  end do
end do
```

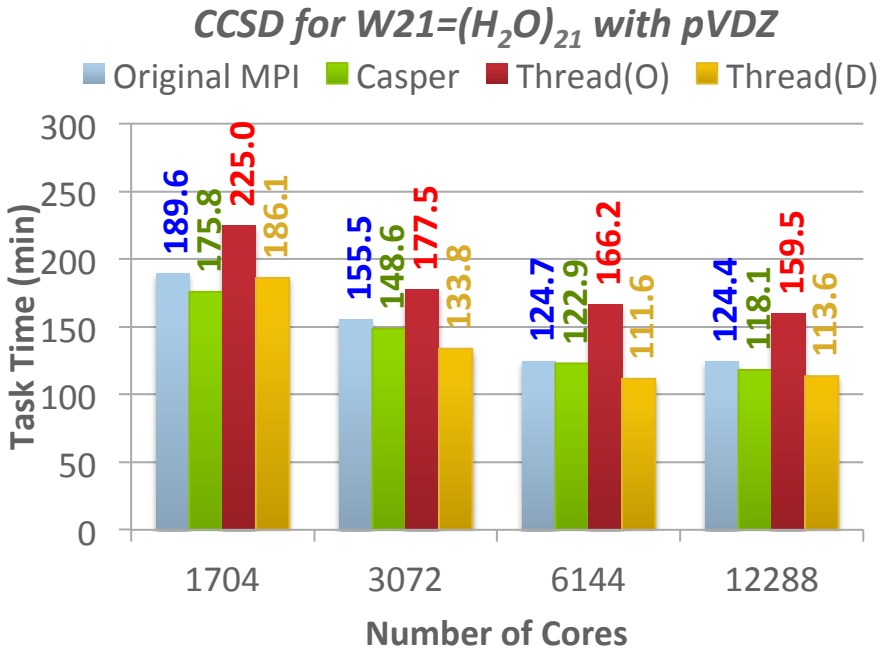
Evaluation 2. NWChem Quantum Chemistry Application (3)

- CCSD for water molecule on Cray XC30

	# COMP	# ASYNC
Original MPI	24	0
Casper	20	4
Thread (O) (with oversubscribed cores)	24	24
Thread (D) (with dedicated cores)	12	12



Casper provides consistent improvement.



The improvement of Casper becomes less with increasing number of cores ?

Evaluation 2. NWChem Quantum Chemistry Application (4)

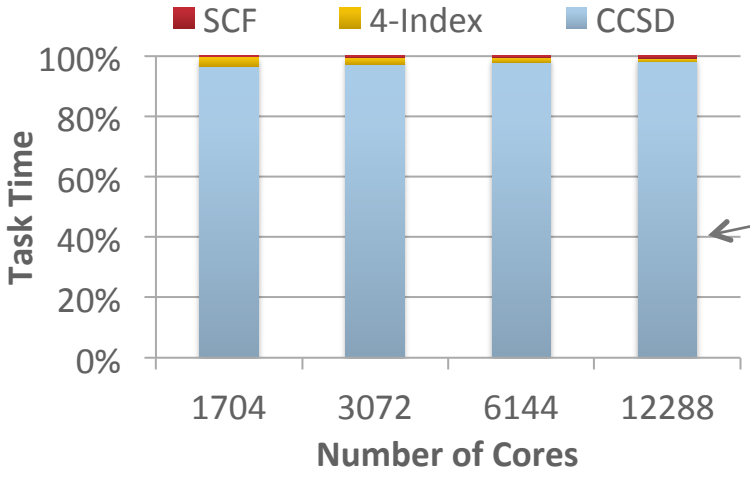
- CCSD profiling

Get-compute-update in CCSD iteration.

```
for each sub block in A, B do
        GET a from A;
        GET b from B;
        DGEMM c=a×b+c;
        ACCUMULATE c to C;
        NXTASK;
done
```

Global Arrays: A, B, C;
Local Buffers : a, b, c;

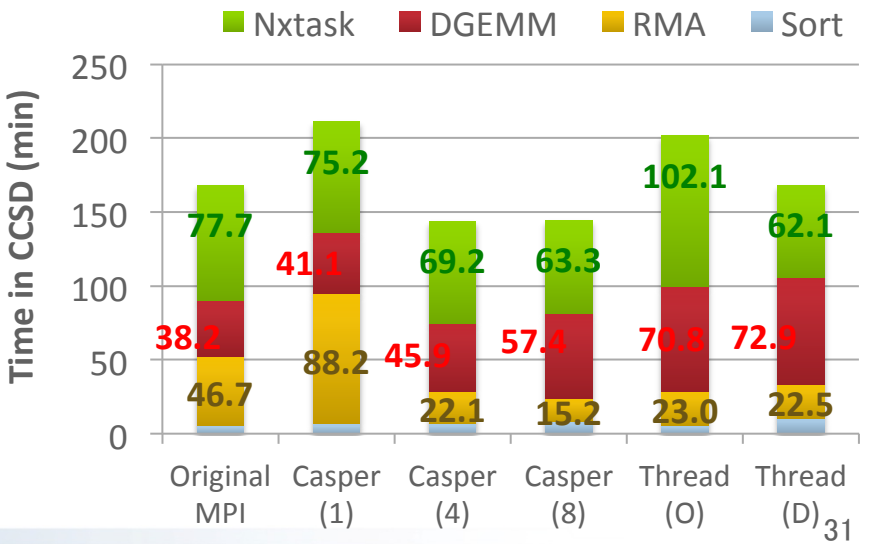
Task internal steps in varying Wn with pVDZ



The CCSD iteration consistently dominates the entire cost of CCSD by close to 90%.

- **The cost of DGEMM is increasing;**
- **The cost of RMA is reducing;**
- **Nxtask takes half of the entire cost.**

CCSD steps in W21-pVDZ with 1704 cores

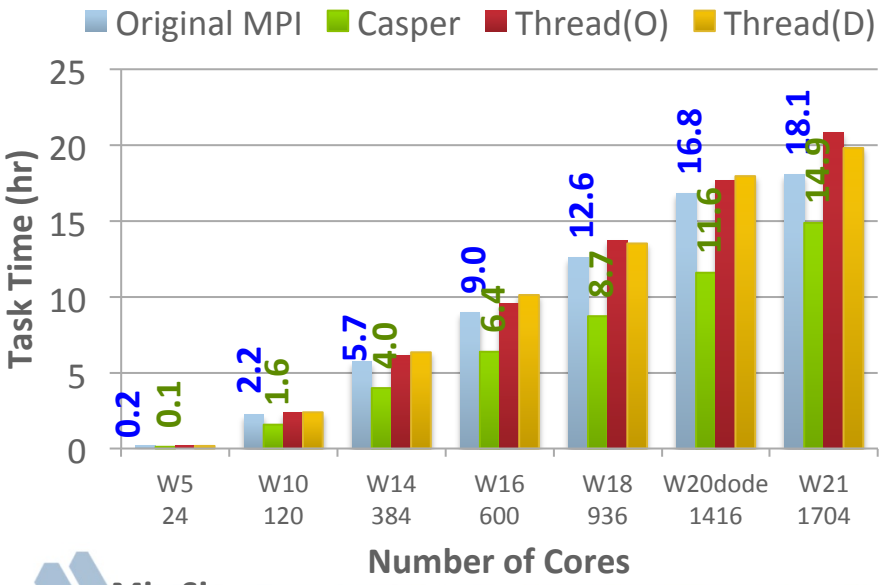


Evaluation 2. NWChem Quantum Chemistry Application (5)

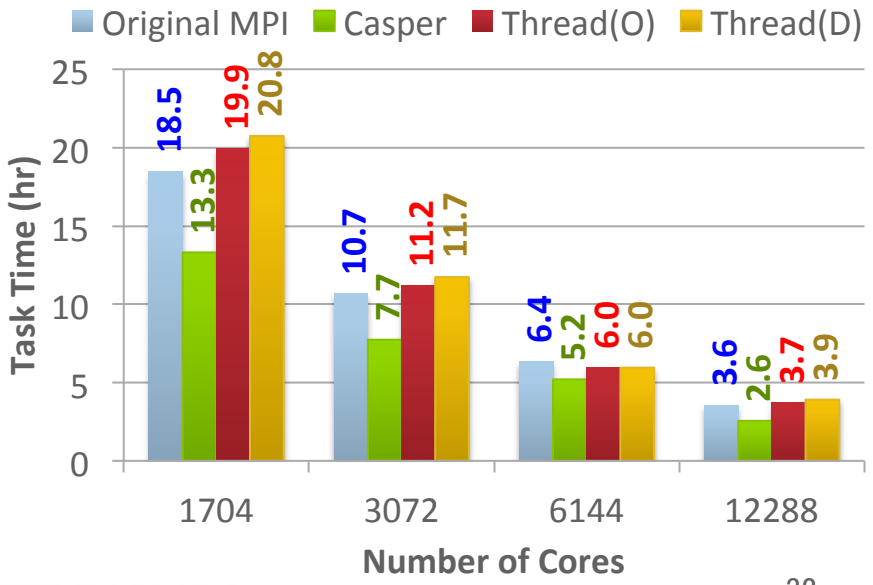
- CCSD(T) for water molecule on Cray XC30.

	# COMP	# ASYNC
Original MPI	24	0
Casper	20	4
Thread (O) (with oversubscribed cores)	24	24
Thread (D) (with dedicated cores)	12	12

CCSD(T) for varying Wn with pVDZ



CCSD(T) for W21=$(H_2O)_{21}$ with pVDZ

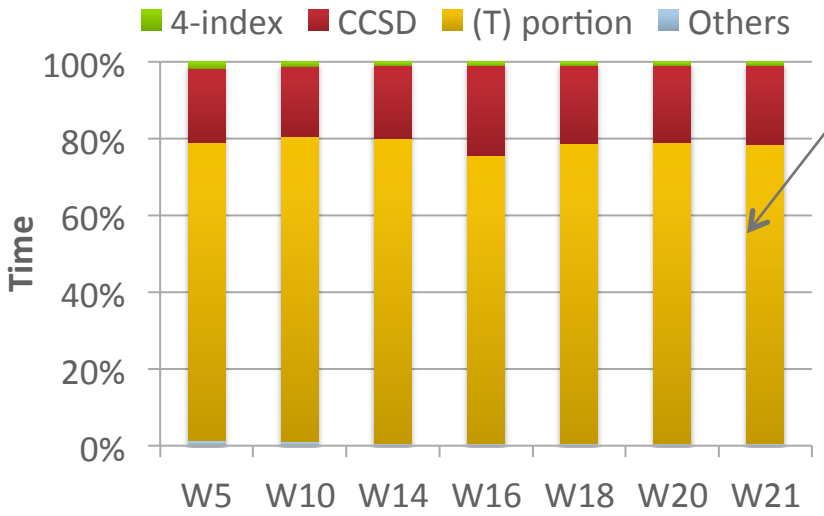


Evaluation 2. NWChem Quantum Chemistry Application (6)

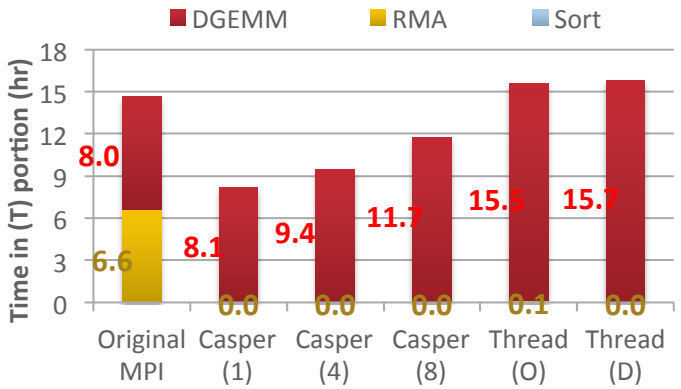
- CCSD(T) profiling

The (T) portion consistently dominates the entire cost by close to 80%.

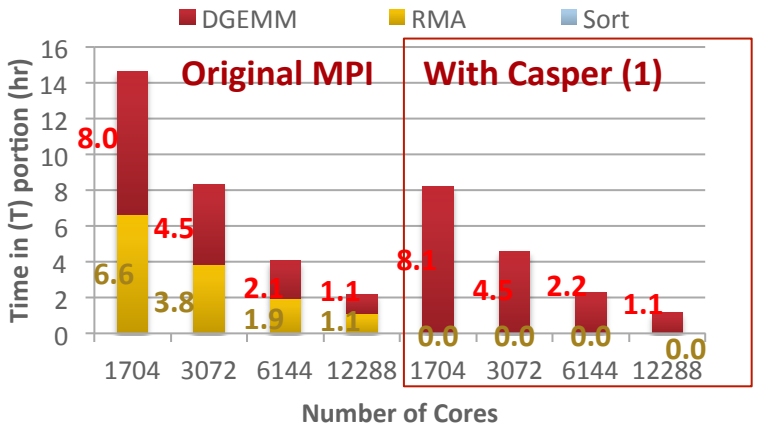
Task internal steps in varying Wn with pVDZ



T portion in W21 –pVDZ with 1704 cores

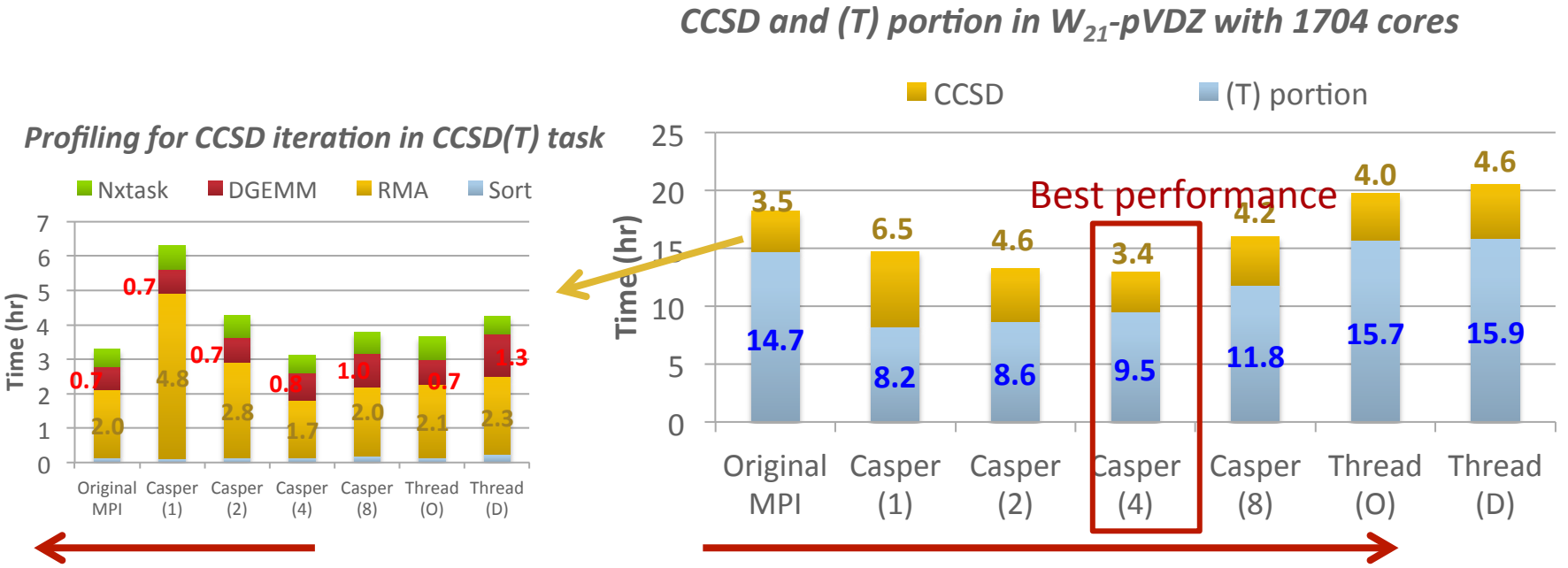


T portion in W21 –pVDZ with varying number of cores



Evaluation 2. NWChem Quantum Chemistry Application (7)

- **CCSD iteration and (T) portion in CCSD(T) task**



With less number of ASYNC cores

- **CCSD gets worse perf because too many RMA operations are handled by only a few ASYNC cores.**

With more number of ASYNC cores

- **Overhead in CCSD(T) increases because of loss of computation cores**

We have to do a trade off in order to deliver the best performance for the entire task…

Argonne NATIONAL LABORATORY

Asynchronous Progress Runtime Adaptation [Ongoing]

Per-window Asynchronous State Management

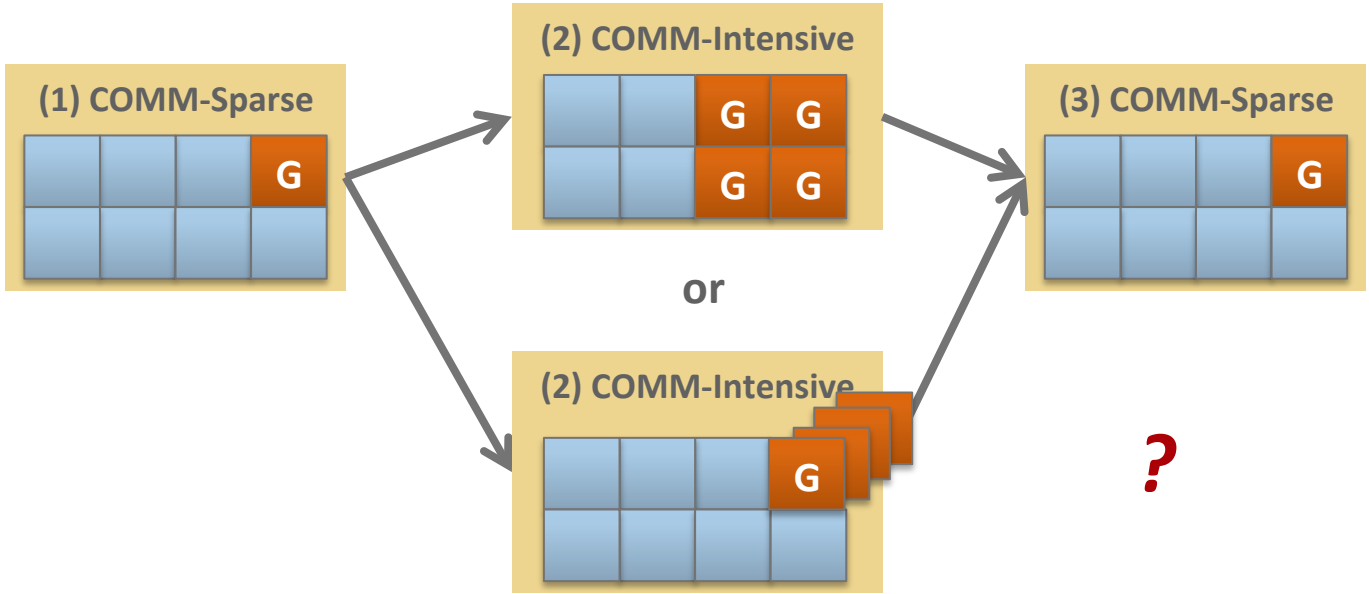
Communication Frequency Monitor

Asynchronous Progress Runtime Adaptation (1)

- **Motivation**
  - Applications always consist of multiple computation phases with different communication characteristics …
    - COMM-Intensive Phase (i.e., CCSD iteration)
      - May not need asynchronous progress
      - Insufficient asynchronous cores may cause communication degradation
    - COMM-Sparse Phase (i.e., (T) portion)
      - Need asynchronous progress
      - Overuse of asynchronous cores may cause computation degradation
  - Can we dynamically change asynchronous progress configuration for different phases ?

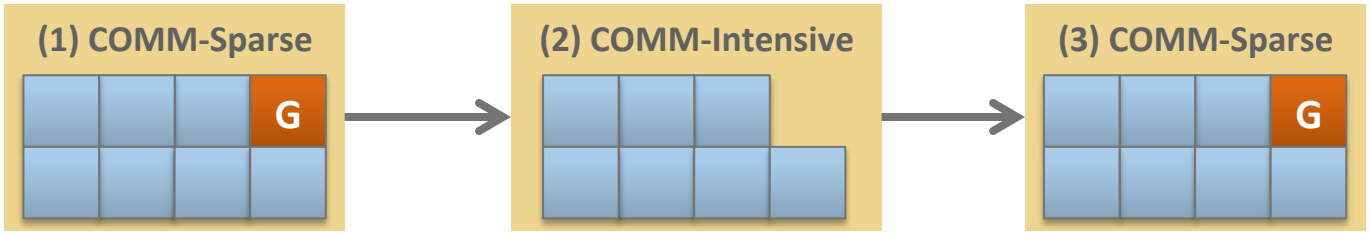
Asynchronous Progress Runtime Adaptation (2)

- Can we transform cores between ASYNC and computing modes ?
  - No, because user has to change the data partitioning
- Can we just add or remove ASYNC processes ?
  - Yes, but may cause significant performance degradation due to core oversubscription

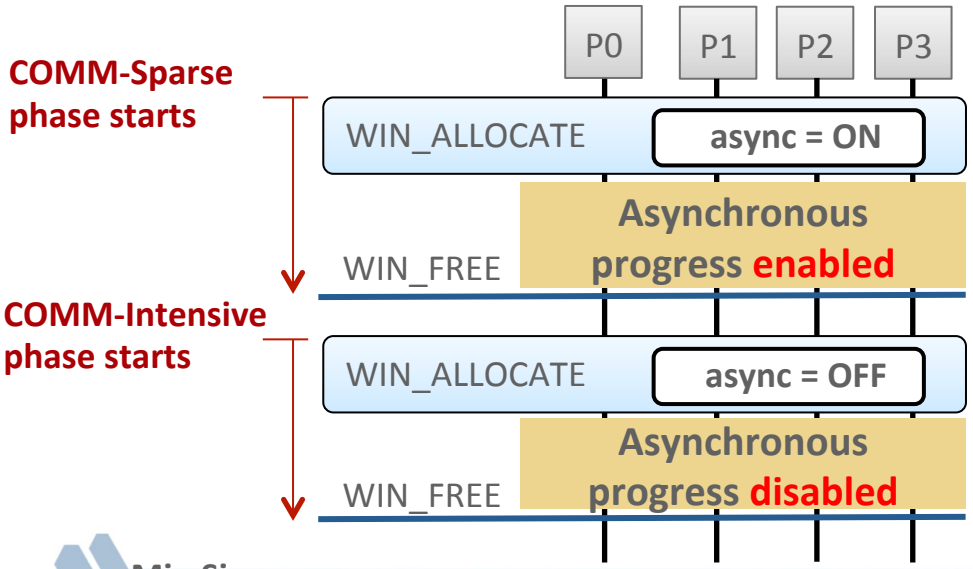


Asynchronous Progress Runtime Adaptation (3)

- **Can we simply turn on /off asynchronous progress ?**
  - **Yes ! And no oversubscription overhead.**



- **Per-window Asynchronous State Management**



User static configuration:

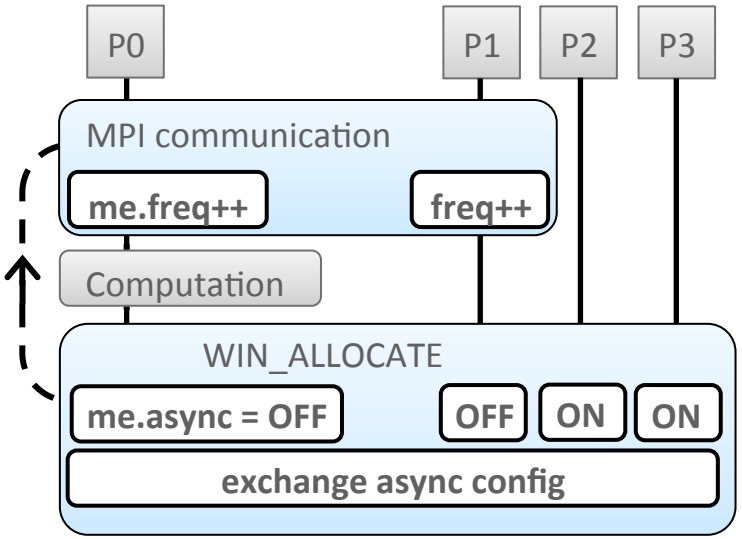
- Pass **async_config info** [ON / OFF] for every window at win_allocate.
- **No correctness concern** since the configuration is consistent through the entire window.

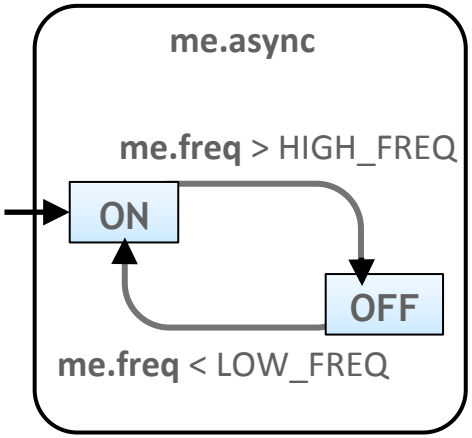
Asynchronous Progress Runtime Adaptation (4)

- **Automatic Adaptation**
  - **Communication Frequency**

$$Freq(t_n) = \frac{CommT_{t_n} - CommT_{t_{n-1}}}{t_n - t_{n-1}}$$

    - **Determines how frequently the communication is performed on the local process.**
    - **If my frequency is high, I don't need asynchronous progress.**
  - **Communication Frequency Monitor**





Current Status:
The frequency judgment is not accurate and stable, how to improve it ?

Next Steps

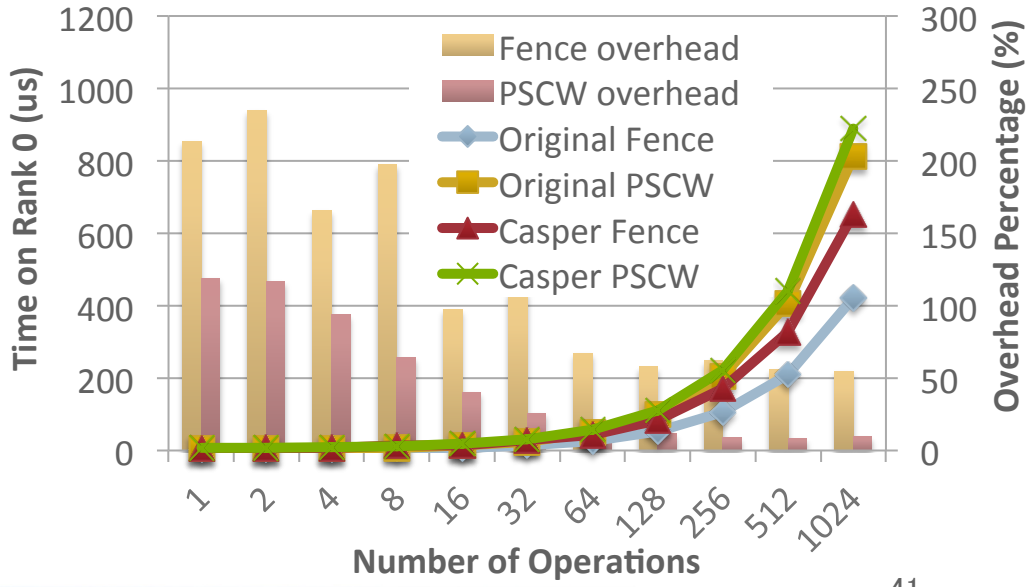
PVAS-based Casper

ULP-based Casper

Restrictions in Current Casper

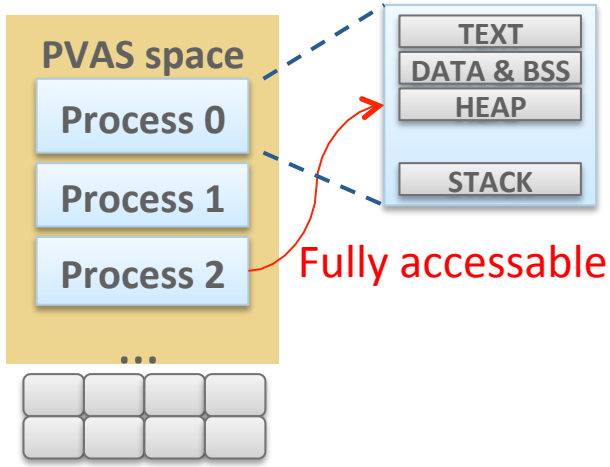
- Limited communication mode
  - Require remote memory accessing from the ghost process to user buffers
  - Only support asynchronous progress on the target side in MPI RMA

- Workaround of MPI Blocking calls
  - Translate Fence / PSCW to passive target mode
  - Additional overhead

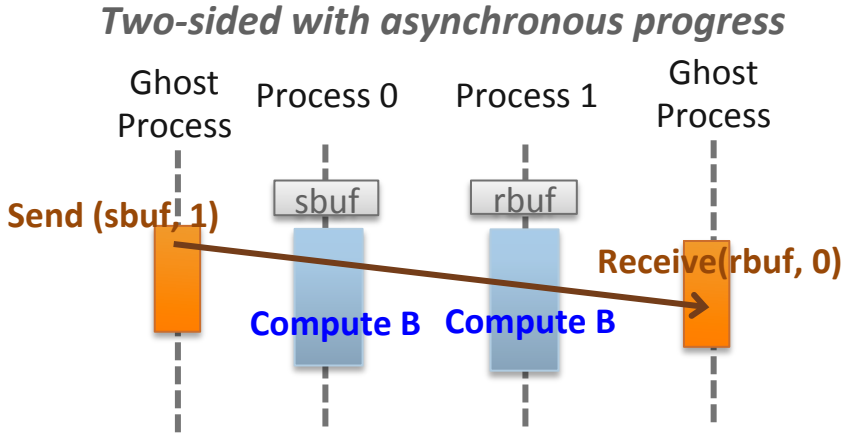


Next Steps (1) : Supporting All Communication Modes

- **P**artitioned **V**irtual **A**ddress **S**pace[1]
  - Proposed by RIKEN, JAPAN
  - An OS process is able to **fully access the memory** address space on any other OS processes inside the same PVAS space.



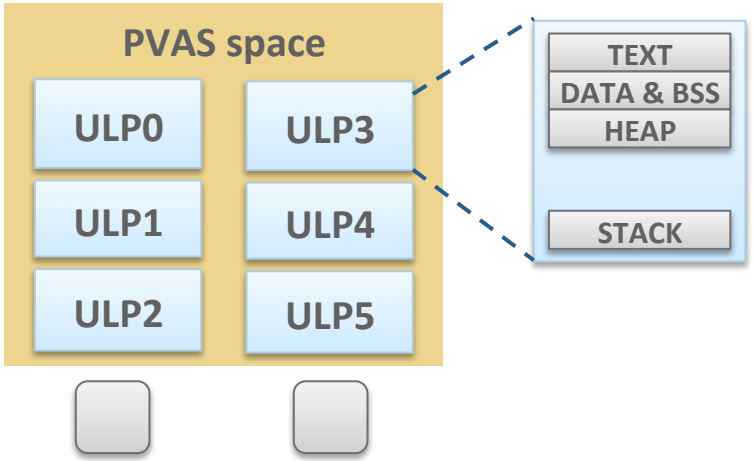
- **PVAS-based Casper**
  - Ghost process is able to fully access user processes in the same PVAS space
  - Support **asynchronous progress for all communication modes**
    - **Origin side in RMA**
    - **Two-sided / collective communication.**

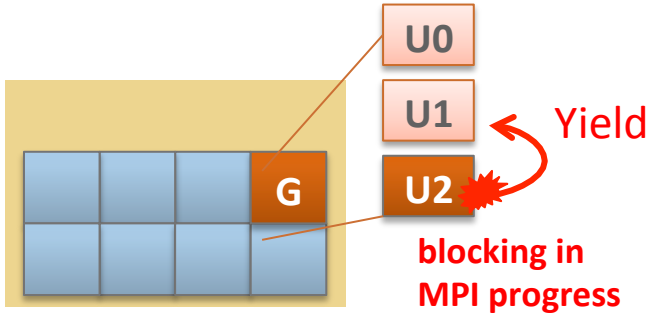


[1] A. Shimada, B. Gerofi, A. Hori, and Y. Ishikawa. Proposing a new task model towards many-core architecture. In Proceedings of the First International Workshop on Many-core Embedded Systems (MES '13). ACM, New York, NY, USA, 45-48.

Next Steps (2) : Supporting Simultaneous Blocking Calls

- **U**ser **L**evel **P**rocess[2]
  - Multiple ULPs on each core, but only one of them is running
  - **Private HEAP & STACK**
    - No multithreading
  - **User controllable scheduling**
    - Yield / Switch
    - Priority setting





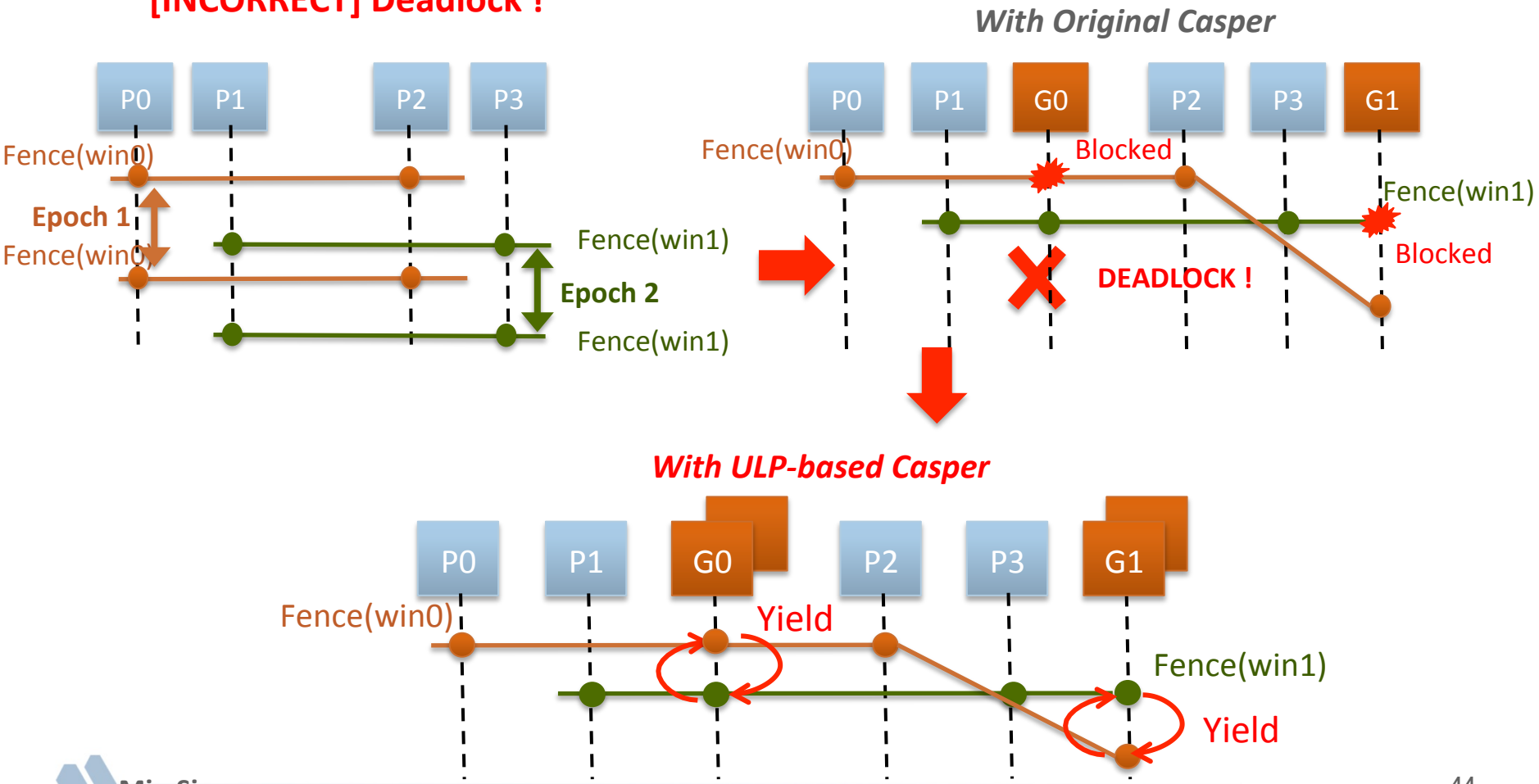
- **ULP-based Casper**
  - Multiple "**Ghost ULPs**" running on one or a few dedicated cores
  - Natively support simultaneous blocking calls (i.e. Fence)

[2] A. Shimada, A. Hori, Y. Ishikawa, and P. Balaji. User-Level Process towards Exascale Systems. IPSJ SIG Technical Report, 2014.

Supporting Simultaneous Blocking Functions in ULP-based Casper

- **Simultaneous fence epochs on disjoint sets of processes sharing the same ghost processes**

[INCORRECT] Deadlock !



Summary

- MPI RMA communication is not truly one-sided
  - Still need asynchronous progress
  - Additional overhead in thread / interrupt-based approaches
- Multi- / Many-Core architectures
  - Number of cores is growing rapidly, some cores are not always busy
- **Casper: a process-based asynchronous progress model**
  - **Dedicating arbitrary number of cores** to ghost processes
  - **Mapping window regions** from user processes to ghost processes
  - **Redirecting all RMA SYNC. & operations** to ghost processes
  - Linking to various MPI implementation through **PMPI transparent redirection**

Download slides: http://sudalab.is.s.u-tokyo.ac.jp/~msi/pdf/casper-seminar-20150423.pdf